Component Integration and Optimization

For High Productivity and Performance

Ken Kennedy
Rice University

http://lacsi.rice.edu/review/slides_2006/

Outline

- Component Integration Systems
  - Support for the maintenance and optimization of component libraries
  - High-productivity languages
- Retargetable High Performance Components
  - Automatic tuning of components for specific computing platforms
  - Design of adaptive components
- Application Drivers from LANL Weapons Program
  - Marmot, Telluride, Ajax programming system (FLAG)
- Previous Projects, Renewed Relevance
  - High-level Java optimization
  - Program Preparation for Heterogenous Computing Environments (e.g., Grids)

Some Previous Accomplishments

- JaMake Java Framework
  - Collaboration with CartaBlanca Project
  - Performs object inlining on arrays of objects
    - Overcomes the cost of using full OO polymorphism
    - Achieved 80% improvement on the LANL Parsek code
    - Critical to CartaBlanca R&D 100 Award
  - Results apply to C++ and Python (and Ajax?)
  - Attracted NSF funding, published 7 refereed papers
- Grid Research
  - Drove performance prediction research
  - Effective performance-model based scheduling
  - VGrADS: NSF ITR (Large)
  - Ideas for Grid in a box
    - Many future supercomputers will have heterogeneous computing components: good scheduling will be critical for performance

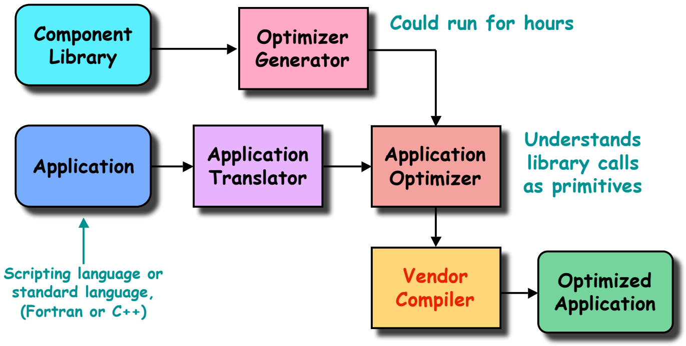
Component Integration

- Supporting Technologies for Component Integration
  - Transformation systems to eliminate overheads due to abstraction
  - Component integration systems to automate specialization
    - Key problem: integration of data structure components with functional components
- Continue Collaborations with LANL Code Projects
  - Marmot
    - Pursue directions in the draft collaboration plan (later)
    - Application of object-oriented optimization strategies (from JaMake, applied to C++ via ROSE infrastructure)
  - New LANL Contact from X Division
    - Hank Alme
- Challenge Applications
  - Export-restricted codes including hydro and transport
  - Representative of ASC code styles

Participants

- LANL Contacts
  - Staff: Hank Alme, Craig Rasmussen
- Rice
  - Faculty/Staff: Ken Kennedy, Bradley Broom*, Zoran Budimlic, Keith Cooper, Arun Chauhan*, Rob Fowler, Guohua Jin, Tim Harvey, Chuck Koelbel, John Mellor-Crummey, Steve Reeves, Linda Torczon
  - Students: Raj Bandyopadhyay, Jason Eckhardt, Mary Fletcher, Alex Grosul, Mack Joyner, Cheryl McCosh, Apan Qasem, Todd Waterman, Anna Youssefi, Rui Zhang, Yuan Zhao
- Tennessee
  - Faculty/Staff: Jack Dongarra, Shirley Moore, Graham Fagg, George Bosilca
  - Students: Haihang You, Jelena Pjesivac-Grbovic, and Jeffery Chen

Telescoping Languages



Telescoping Language Advantages

- Optimized script compilation times can be reasonable
  - Investment in library analysis speeds script optimization
- High-level optimizations possible
  - Exploit library designer's knowledge of routine properties
  - Specialize library routines during optimizer generation to exploit expected calling sequences
    - Apply high-level transformations based on identities
    - Factor and/or fuse library primitives as appropriate
- User retains substantive control over performance
  - Mature code can be built into a library, annotated with properties to aid optimization and fed to library compiler
- Reliability can be improved
  - No hand coding to context

Component Integration System

- Component integration systems are important productivity tools
- Programs constructed from them can be slow
  - No context-based code improvements can be applied
  - Component crossing overheads are high
    - Result: heavyweight components, insufficient separation of concerns
- Claim: Telescoping languages can address this problem
  - Can be applied to construct component integration systems that yield high-performance applications
  - Can make components usable in contexts that have been previously considered impractical
- ASC Relevance
  - Component-based software is critical for productivity and reliability
  - Performance must be high for software to be usable
  - Useful to prototype in high-productivity language (Python, Matlab)

Component Integration Challenge

- Integration of different component libraries that
  - Implement data structures (e.g., sparse matrices)
  - Implement functions on data structures (e.g., linear algebra)
- Problem: Performance
  - High function overhead for data structure access (frequently invoked)
  - Need optimization for special contexts
    - e.g., invocation in loops
- Claim: Telescoping languages can handle this well
  - Advance generation of specialized entries
  - Transformation pass to perform substitution

What We Have Done

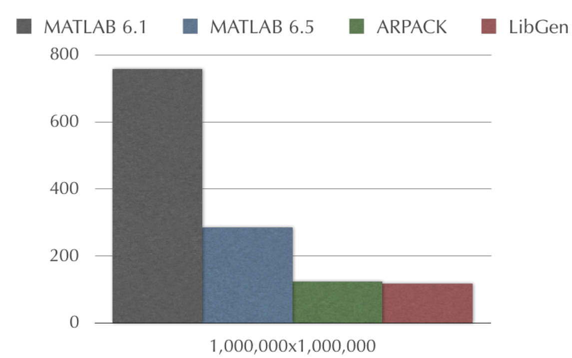
- Developed base-language compiler technology
  - Type inference: Key to generation of C or Fortran from Matlab, S, or Python
    - Useful even if C++ or Fortran is your scripting language
- Conducted preliminary studies
  - Matlab SP (Signal Processing), LibGen (library generation)
    - Six papers, one Ph.D., two Master's
  - R compilation (funded separately by DOD)
- Demonstrated benefits of telescoping languages as component integration system (via LibGen)
- Developed strategy for generalized data structures
  - Including addition of parallelism to scripting languages (funded by ST-HEC program from NSF/DARPA)
- Met with Marmot team to explore collaboration opportunities
- Begun examination of applicability to codes written using Ajax Programming System (e.g., FLAG)

Library Generator (LibGen)

- ARPACK
  - Prof Dan Sorensen (Rice CAAM) maintains ARPACK, a large-scale eigenvalue solver
- Methodology
  - He prototypes the algorithms in Matlab, then generates 8 variants in Fortran by hand:
    - {Real, Complex} x {Symmetric, Nonsymmetric} x {Single,Double}
    - Dense vs Sparse handled by special interface
- Could this hand generation step be eliminated?
  - Answer: YES
  - Key technology: Constraint-based type inference
    - Polynomial time algorithm to compute type jump functions

      Map input types to variable types

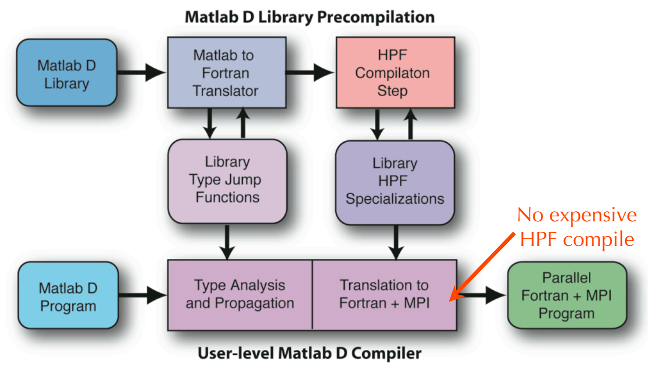
LibGen Performance



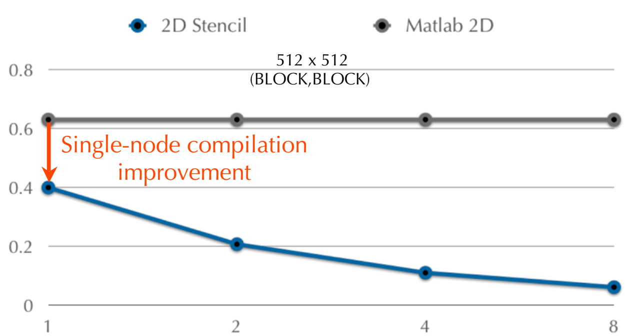
Parallelism in Scripting Languages

- Tennessee Approach
  - Global arrays resident on parallel ScaLAPACK server
    - Operations executed local to data
  - Accessible from Matlab, Python, Mathematica clients
    - R should be an easy extension
  - Working now
- Rice Approach
  - Support for multiple distributions
    - Standard plus user-defined
  - Compilation to Fortran 90 + MPI
    - Runs on back end server
  - Telescoping languages + HPF technology for specialization
    - Specialize each operation for each distribution
    - Specialize communication for each distribution pair
  - Funded by NSF ST-HEC (DARPA HPCS)

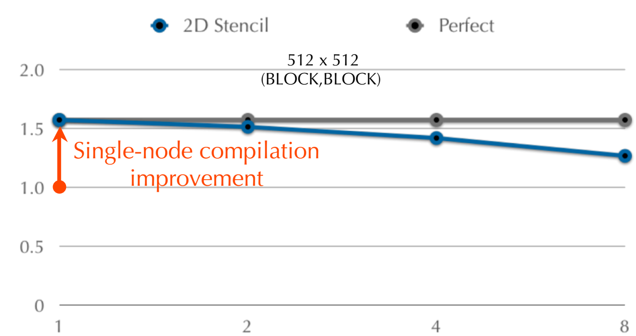
Rice Parallel Matlab



Performance: 2D Stencil



Parallel Efficiency: 2D Stencil



LACSI Interactions

- Priorities and Strategies Meetings
  - Inputs from Steven Lee and Ken Koch led to direction change
- Attended Workshops
  - Common Component Architecture, LACSI Symposium 2002
  - Initial Components Workshop (April 16-17, 2003)
- Discussions with Marmot Group
  - Monterrey Methods Workshop (March 16-18, 2004)
  - Components Workshop at LANL (June 24, 2004)
    - Developed an outline plan for collaboration
  - Additional meetings during LACSI visit Oct 31-Nov 3, 2005
- Meetings with Code Performance Team (Alme)
  - October-November 2005 visit, December 2005
    - Leading to focus on Ajax (Scott Runnels)

LACSI

What We Plan to Do

- Seek (and solve) component integration challenge problem
  - Emphasis on efficiency of frequent component-crossing
    - Integration of data structure and function
- Explore opportunities in other ASC codes
  - Initiating new project to explore improvements in Ajax programming system (on FLAG)
    - Telescoping languages and object-oriented optimizations
- Continue interactions with Marmot Project
  - Goal: build tools to help them on their second or third iteration
- Explore application of parallel Matlab and R to V&V codes from D1 (Dave Higdon)
- Relevance to ASC
  - Success will make it easier to use modern component-based software development strategies in ASC codes
    - Without sacrificing performance

LACSI

Specific Topics

- Specialization Strategies
  - Specialized handling of multiple materials in cells
  - Compiler-based specialization to sparse data structures
  - Combined telescoping languages and dynamic code selection
    - Optimization by limited computation reorganization
- Improved Translation within Ajax
- Tools for Preoptimization of Libraries
  - Pre-specialization of library codes to expected calling contexts
  - Potential source of components: Trillinos
- Mining of Traditional Applications
  - Construction of libraries for inclusion in domain languages
- Rapid Prototyping Support
  - Compilation of scripting languages (Python, Matlab) to Fortran/C

LACSI

Automatic Component Tuning

- Participants: Four Groups within LACSI
  - Tennessee: Jack Dongarra
    - Collaboration with LLNL ROSE Group (Dan Quinlan, Qing Yi)
  - Rice: Ken Kennedy and John Mellor Crummey
    - Students Apan Qasem and Yuan Zhao
    - Also collaborating with ROSE Group
  - Rice: Keith Cooper, Devika Subramanian, and Linda Torczon
    - Students Todd Waterman and Alex Grosul

LACSI

Automatic Component Tuning

- Goal: Pretune components for high performance on different computing platforms (in advance)
  - Models: ATLAS, FFTW, UHFFT
  - Generate tuned versions automatically
- Strategy: View as giant optimization problem with code running time as objective function
  - For each critical loop nest:
    - Parameterize the search space
    - Prune using compiler models based on static analysis
    - Employ heuristic search to find optimal point and generate optimal code version
  - Typical optimizations:
    - Loop blocking, unroll, unroll-and-jam, loop fusion, storage reduction, optimization of target compiler settings, inlining, optimization of function decomposition

New Autotuning Work

- Combination of three optimizations (student: Apan Qasem)
  - Cache tiling, register blocking, and loop fusion (new)
- Loop fusion
  - Critical for performance, particularly on Fortran 90 codes
    - Array assignments are “scalarized” in multiple dimensions
    - Fusion can dramatically enhance memory performance
  - Problems:
    - too much fusion can lead to conflict misses
    - fusion and tiling interactions can degrade performance
  - Strategy:
    - Strategy: construct combined model that predicts “effective cache size” (fewer than 2.5% conflict misses) then fuse and tune to that size
    - Advantage: many fewer evaluations, basically same performance

Automatic Tuning

- Successes
  - Experimental infrastructure
    - LoopTool, MSCP, ATLAS2, CODELAB
  - Large-scale experiments
  - Principles demonstrated
    - Effectiveness of heuristic search (including parallel search)
    - Importance of search-space pruning using compiler models
  - Papers published
    - Ten refereed publications and one technical report (see web site)
  - Established an Autotuning Community
    - LACSI Workshop 2005
- Relevance
  - Dramatically increases productivity of scientific programming
- Connections to ASC
  - Sweep3D (1.3x on Alpha), ASC performance group, Marmot, Truchas

Summary

- Component integration languages and frameworks
  - High Level: Matlab, S, Python plus component libraries
  - Low Level: C, C++, Fortran
- Compilation technology
  - Type inferencing to drive translation to C or Fortran
  - Telescoping languages to pre-optimize libraries
  - Parallelism in scripting languages
    - Parallelism based on distribution
- Component Autotuning
  - Goal: ATLAS-style automatic tuning for generalized applications, UHFFT-style automatic tuning for decomposable (library) components
  - Exploring heuristic search and static search-space pruning
- Technology Transfer
  - Focus component integration on problems arising from ASC code projects
  - Automatic tuning applicable to general languages